Protocol for Asynchronous, Reliable, Secure and Efficient Consensus (PARSEC)

Pierre Chevalier, Bartłomiej Kamiński, Fraser Hutchison, Qi Ma,
Andreas Fackler, Spandan Sharma *

May 24, 2018

Abstract

In this paper we present an asynchronous algorithm for reaching consensus in the presence of Byzantine faults. We prove the algorithm's correctness provided that less than a third of participating nodes are faulty.

Keywords: asynchronous, byzantine, consensus, distributed

1 Introduction

This paper presents a new byzantine fault tolerant, asynchronous consensus algorithm. Like [1], it has complete asynchrony, no leaders, no round robin, no proof-of-work and reaches eventual consensus with probability one. However unlike [1], it does not only provide high speed in the absence of faults, but also in their presence. It is also fully open, and a GPLv3 implementation written in Rust will be made available in the near future.

Like [4], this algorithm is built by composing a number of good ideas present in the literature. The gossip protocol is used to allow efficient communication between nodes, as in [1] and [5]. Propagating a message, and indeed, reaching consensus only costs $O(N \log N)$ communications and $O(\log N)$ stages.

The general problem of reaching Byzantine agreement on any value is reduced to the simpler problem of reaching binary Byzantine agreement on the nodes participating in each decision. This allows us to reuse the elegant binary Byzantine agreement protocol described in [2] after adapting it to the gossip protocol.

Finally, the need for a trusted leader or a trusted setup phase implied in [2] is removed by porting the key ideas from [3] to an asynchronous setting.

The resulting algorithm is a Protocol for Asynchronous, Reliable, Secure and Efficient Consensus.

PARSEC is a key building block of the SAFE Network, an ethical decentralized network of data and applications providing Secure Access For Everyone.

* MaidSafe Ltd., emails: firstname.lastname@maidsafe.net

2 The algorithm description

2.1 The network model

We assume the network to be a set $\mathcal{N}$ of N instances of the algorithm communicating via asynchronous connections. By "asynchronous" we mean that messages are delivered eventually, but can be arbitrarily delayed, so that it is impossible to tell whether an instance has failed by completely stopping, or there is just a delay in message delivery. We allow a possibility of up to t Byzantine (arbitrary) failures, where $3t < N$. We will call the instances that haven't failed *correct* or *honest*, and the failing instances *faulty* or *malicious* - as Byzantine failure model allows for malicious behaviour and collaboration.

We will refer to any set of instances containing more than $\frac{2}{3}N$ of them as a *supermajority*.

2.2 Data structures

A node executing the algorithm keeps two data structures: a *gossip graph* and an ordered set of *blocks*. The vertices of the gossip graph, called *gossip events*, contain the following fields:

- Payload - data the node wants to pass to other nodes
- Self-parent (optional) - a cryptographic hash of another gossip event created by the same node
- Other-parent (optional) - a hash of another gossip event created by some other node
- Cause - cause of creation for this event; can be *request*, *response* or *observation*
- Creator ID - the public key of the event's creator
- Signature - a cryptographic signature of the above fields

The self-parent and other-parent are always present, except for the first events created by respective nodes, as there are no parent events to be referred to in such cases. Other-parent is also absent in events created because of an observation - because there is no gossip partner in such a case.

The blocks in the ordered set are network events signed by a subset of the nodes in the network. This set is the output of the algorithm, and represents an order of network events that all nodes agree upon.

Let us also define a few useful terms regarding the gossip graph for future use.

Definition 2.1. We say that event A is an *ancestor* of event B iff: $A = B$, or A is an ancestor of B's self-parent, or A is an ancestor of B's other-parent.

Definition 2.2. We say that event A is a *descendant* of event B iff B is an ancestor of A.

Definition 2.3. We say that event A is a *strict ancestor/descendant* of event B iff A is an ancestor/descendant of B and $A \neq B$.

Following Swirlds[1], we also define additional two useful notions:

Definition 2.4. An event A is said to *see* an event B iff B is an ancestor of A, and there doesn't exist any pair of events by B's creator B_1, B_2, such that B_1 and B_2 are ancestors of A, but B_1 is neither an ancestor nor a descendant of B_2 (see fig. 1). We call a situation in which such a pair exists a *fork*.

Definition 2.5. An event A is said to *strongly see* an event B iff A sees a set of events created by a supermajority of nodes in the system that all see B (see fig. 2).

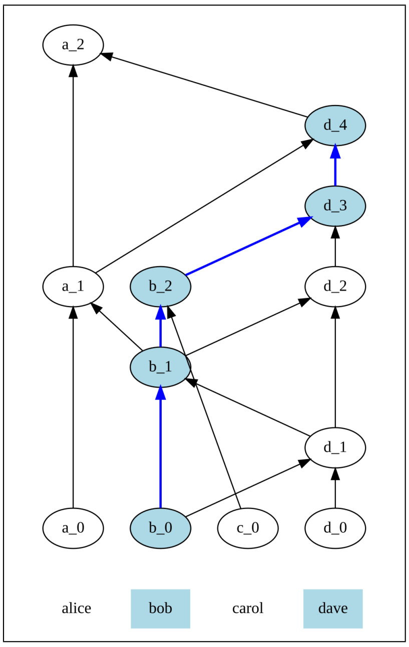

Figure 1: d_4 sees b_0: b_0 is its ancestor and there are no forks

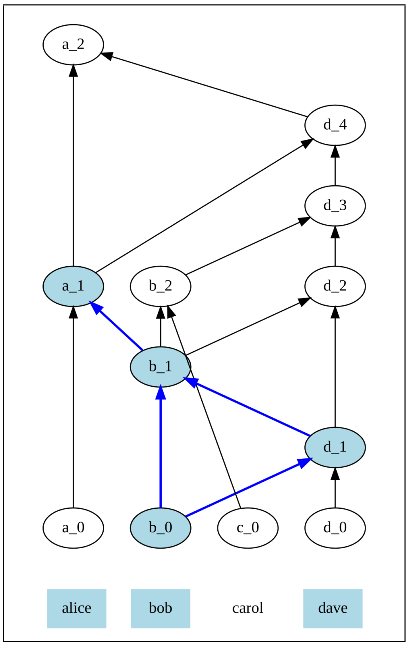

Figure 2: a_1 strongly sees b_0: it sees itself, b_1 and d_1, which are a supermajority and all see b_0

2.3 General overview of the algorithm

The nodes execute two main steps in an infinite loop:

- Synchronise the gossip graph with another random node
- Determine whether any new blocks should be appended to the ordered set

2.3.1 Synchronisation

This step is responsible for building the gossip graph and spreading information around the network. Nodes continually make random calls, called *sync requests*, to other nodes and exchange information about the graph, so that all correct nodes end up with the same data in their graphs. The hashes and signatures in gossip events make sure that malicious nodes won't be able to tamper with any part of the graph.

Whenever a node receives a sync request, it creates a new gossip event and sends a sync response back. The self-parent of this event is the hash of the last gossip event created by the recipient, and the other-parent is the hash of the last event created by the sender (which the recipient learns about from the exchange). The recipient of the sync response also creates a new event with analogous parents. Both events created also store the reason for which they were created (whether due to a request, or a response).

If the recipient of a request/response believes it knows a network event that should be appended as the next one in the chain, it records its vote as the payload of the newly created event. The other nodes will learn of this vote during subsequent sync exchanges made by its creator.

2.3.2 Determining order

During this step, a node analyses the graph, counts the votes and decides which block should become the next one. This step is a complex one and so it is described in detail in a separate subsection below.

2.4 Calculating the order

To be able to order blocks, we need first to have some blocks that can be ordered.

As mentioned above, the gossip events may contain votes for network events. A gossip event which sees events created by a supermajority of nodes that contain votes for a given network event is said to *see a valid block*, and we will call such gossip events *block-votes*. The first gossip event which strongly sees block-votes created by a supermajority of nodes is said to be an *observer*. The block-votes don't need to see the same valid block - in fact, it is the case when they see different valid blocks that is the most interesting. However, they do have to only refer to blocks that haven't been appended to the ordered set yet.

An observer implicitly carries a list of N *meta-votes*. Every meta-vote is just a binary value denoting whether a corresponding node's block-vote is to be taken into account when determining the order. An observer meta-votes *true* on a node if it can strongly see a block-vote by that node. Every node is being meta-voted on, hence there are N meta-votes, and since an observer strongly sees a supermajority of block-votes, by definition, at least $\frac{2}{3}N$ of them are *true*.

Meta-votes reduce the problem of Byzantine agreement about the order to that of binary Byzantine agreement, which has been solved previously[2].

The algorithm described in [2] has some shortcomings, though, the most significant of which is the need for a *common coin*, a primitive which may require synchronicity and/or a trusted third party for efficient creation or setup. The algorithm presented here works without such a requirement.

2.4.1 Binary agreement

For the sake of simplicity, we will define the algorithm in terms of deciding a single *meta-election* - that is, deciding whether or not to take a single node's opinion into account when trying to choose a single new block. We can view a meta-election for node X with latest agreed block B as a function on a subset

$H_{X,B}$ of the gossip graph G, which is the set of all events that are descendants of any observer of this meta-election:

$$\mathsf{meta\_election}_{X,B} : H_{X,B} \to \{0, 1, \bot\}$$

The $\bot$ value means that the result has not been decided yet at this point in the graph.

In order to calculate the meta-election value for events in $H_{X,B}$, we will need to calculate a few helper values as well:

- stage - a counter denoting the calculation stage
- estimates - a set of one or two values estimating the final result
- bin_values - a helper set of binary values
- aux - a helper binary value

stage is an integer value which represents the stage of the protocol we are considering when looking at a specific gossip event. A number is associated with each gossip event, such that the stage of the observers is always 0. The stage of any other gossip event is either the stage of its self-parent, or the stage of its self-parent plus one under specific conditions. The exact conditions under which the stage is incremented will be described later in more details. Other variables such as estimates, bin_values and aux all depend on the stage.

estimates is a set of binary values that represent the perceived opinion(s) of the creator of any gossip event on the outcome of a meta-election. The estimate of an observer is the set containing just its own meta-vote. The estimate of any subsequent gossip event can be a different set as described below.

If the estimates for an event's self-parent contain a single value v, and that event sees more than $\frac{N}{3}$ events with $\neg v$ in their estimates (which means that at least one honest node estimated $\neg v$), this opposite value gets added to its own estimates (so it will contain both true and false).

$$\mathsf{est} : H_{X,B} \to 2^{\{0,1\}}$$

$$\mathsf{est}(e) = \begin{cases} \{v\} & \text{if there exists an ancestor } d \text{ of } e \\ & \text{such that } v = \mathsf{meta\_election}(d) \neq \bot \\ \{w\} & \text{if } e \text{ is an observer with meta-vote } w \\ \{0,1\} & \text{if } \mathsf{est}(\mathsf{self\_par}(e)) = \{v\} \\ & \text{and } e \text{ sees} \geq \frac{N}{3} \text{ events } x \\ & \text{by different nodes such that} \\ & \mathsf{stage}(x) = \mathsf{stage}(e) \text{ and } \neg v \in \mathsf{est}(x) \\ \mathsf{next\_est}(\mathsf{self\_par}(e)) & \text{if } \mathsf{stage}(e) > \mathsf{stage}(\mathsf{self\_par}(e)) \\ \mathsf{est}(\mathsf{self\_par}(e)) & \text{otherwise} \end{cases}$$

self_par(e) denotes e's self-parent, and next_est and will be defined later, once we have defined more values related to the events.

Once an event can see a supermajority of events by different nodes which agree in their estimates, this agreed estimate becomes an element of this event's bin_values. This set serves to validate values proposed by other nodes - if they propose something we don't have in $\mathsf{bin\_values}$, we will reject it, as we have no way to ensure its validity.

$$\mathsf{bv} : H_{X,B} \to 2^{\{0,1\}}$$

$$\begin{aligned}\mathsf{bv}(e) = \{v : \ & \text{there exist} > \tfrac{2}{3}N \text{ events } x \\ & \text{by different nodes such that} \\ & e \text{ sees } x \text{ and } \mathsf{stage}(e) = \mathsf{stage}(x) \text{ and } v \in \mathsf{est}(x)\}\end{aligned}$$

If an event's parent has an empty aux value, and the event itself has non-empty $\mathsf{bin\_values}$, it can propose a value to be agreed. This proposing is realised by having a non-empty aux value. If $\mathsf{bin\_values}$ contains just one value, this value becomes the aux value; otherwise, we can pick an arbitrary value, so we will pick true. If the parent's value isn't empty, it becomes our value as well.

$$\mathsf{aux} : H_{X,B} \to \{0, 1, \bot\}$$

$$\mathsf{aux}(e) = \begin{cases} v & \text{if there exists an ancestor } d \text{ of } e \\ & \text{such that } v = \mathsf{meta\_election}(d) \neq \bot \\ \bot & \text{if } \mathsf{bv}(e) = \varnothing \\ w & \text{if } \mathsf{bv}(e) = \{w\} \\ & \text{and } \mathsf{aux}(\mathsf{self\_par}(e)) = \bot \\ 1 & \text{if } \mathsf{bv}(e) = \{0,1\} \\ & \text{and } \mathsf{aux}(\mathsf{self\_par}(e)) = \bot \\ \mathsf{aux}(\mathsf{self\_par}(e)) & \text{if } \mathsf{aux}(\mathsf{self\_par}(e)) \neq \bot \end{cases}$$

Whenever an event sees a supermajority of events with valid aux values, we perform the gradient leadership based concrete coin protocol, which will lead either to deciding the final agreed value, or updating the estimates and moving to the next stage.

First, let us define some helper functions:

$$\mathsf{supermajority\_valid\_aux} : H_{X,B} \to \{0, 1\}$$

$$\begin{aligned}\mathsf{supermajority\_valid\_aux}(e) = \ & e \text{ sees a supermajority} \\ & \text{of events } x \text{ by different nodes} \\ & \text{such that } \mathsf{stage}(x) = \mathsf{stage}(e) \\ & \text{and } \mathsf{aux}(x) \in \mathsf{bv}(e)\end{aligned}$$

$$\mathsf{count\_aux} : H_{X,B} \times \{0, 1\} \to \mathbb{N}$$

$$\begin{aligned}\mathsf{count\_aux}(e, v) = \ & \text{number of events } x \text{ by different nodes such that} \\ & e \text{ sees } x \text{ and } \mathsf{stage}(x) = \mathsf{stage}(e) \\ & \text{and } \mathsf{aux}(x) \in \mathsf{bv}(e) \text{ and } \mathsf{aux}(x) = v\end{aligned}$$

Now we can define how to determine a decided value:

$$\mathsf{meta\_election} : H_{X,B} \to \{0, 1, \bot\}$$

$$\mathsf{meta\_election}(e) = \begin{cases} v & \text{if there exists an ancestor } d \text{ of } e \\ & \text{such that } v = \mathsf{meta\_election}(d) \neq \bot \\ 1 & \text{if } \mathsf{stage}(e) \equiv 0 \pmod 3 \\ & \text{and } \mathsf{count\_aux}(e, 1) > \frac{2}{3}N \\ 0 & \text{if } \mathsf{stage}(e) \equiv 1 \pmod 3 \\ & \text{and } \mathsf{count\_aux}(e, 0) > \frac{2}{3}N \\ \bot & \text{otherwise} \end{cases}$$

If an event sees a supermajority of valid aux values, but isn't able to decide, the next event will mark the beginning of the next stage of the algorithm. This lets us finally define stage:

$$\mathsf{stage} : H_{X,B} \to \mathbb{N}$$

$$\mathsf{stage}(e) = \begin{cases} 0 & \text{if } e \text{ is an observer} \\ \mathsf{next\_stage}(\mathsf{self\_par}(e)) & \text{otherwise} \end{cases}$$

$$\mathsf{next\_stage} : H_{X,B} \to \mathbb{N}$$

$$\mathsf{next\_stage}(e) = \mathsf{stage}(e) + \begin{cases} 1 & \text{if } \mathsf{supermajority\_valid\_aux}(e) \\ & \text{and } \mathsf{next\_est}(e) \neq \bot \\ 0 & \text{otherwise} \end{cases}$$

We will also define two auxiliary values derived from stage - step and round. They will be more convenient than stage in the description below.

$$\mathsf{round} : H_{X,B} \to \mathbb{N}$$

$$\mathsf{round}(e) = \mathsf{stage}(e)/3$$

$$\mathsf{step} : H_{X,B} \to \mathbb{N}$$

$$\mathsf{step}(e) = \mathsf{stage}(e) - \mathsf{round}(e) \times 3$$

Under this definition, the sequential stages $0, 1, 2, 3, ...$ will translate to round 0, step 0; round 0, step 1; round 0, step 2; round 1, step 0; etc.

If we don't decide in a stage, we need new estimates for the next one. This is being taken care of by the three-step concrete coin protocol briefly mentioned before:

- In any round in step 0, we decide true if we see a supermajority of true aux values. If we see a supermajority of false values, we update the estimate to false. If we don't see any supermajority, we estimate true in the next step.

- In any round in step 1, we proceed analogously to above, but in the opposite way: if we see a supermajority of false aux values, we decide false; if we see a supermajority of true values, we estimate true; if we don't see any supermajority, estimate false.

- Step 2 of any round is a genuinely flipped concrete coin step. If we see a supermajority of agreeing aux values, we update our estimate to that value, otherwise we flip a concrete coin (described below) and update our estimate to that. We never decide in a coin step.

How first two points work with regards to deciding can be seen in the definition of meta_election above. To calculate new estimates, we will define a next_est function (which appeared already in the definition of est):

$$\mathsf{est\_true} : H_{X,B} \to \{0,1\}$$

$$\begin{aligned}\mathsf{est\_true} = \; & [(\mathsf{step}(e) = 1 \text{ or } \mathsf{step}(e) = 2) \text{ and } \mathsf{count\_aux}(e,1) > \tfrac{2}{3}N] \\ & \text{or } (\mathsf{step}(e) = 0 \text{ and } \mathsf{count\_aux}(e,0) \leq \tfrac{2}{3}N \\ & \text{and } \mathsf{meta\_election}(e) = \bot)\end{aligned}$$

$$\mathsf{est\_false} : H_{X,B} \to \{0,1\}$$

$$\begin{aligned}\mathsf{est\_false} = \; & [(\mathsf{step}(e) = 0 \text{ or } \mathsf{step}(e) = 2) \text{ and } \mathsf{count\_aux}(e,0) > \tfrac{2}{3}N] \\ & \text{or } (\mathsf{step}(e) = 1 \text{ and } \mathsf{count\_aux}(e,1) \leq \tfrac{2}{3}N \\ & \text{and } \mathsf{meta\_election}(e) = \bot)\end{aligned}$$

$$\mathsf{next\_est} : H_{X,B} \to 2^{\{0,1\}} \cup \{\bot\}$$

$$\mathsf{next\_est}(e) = \begin{cases} \{1\} & \text{if } \mathsf{est\_true}(e) \\ \{0\} & \text{if } \mathsf{est\_false}(e) \\ \{\mathsf{coin\_flip}(e)\} & \text{if } \mathsf{coin\_flip}(e) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

coin_flip is a function that gives the result of the concrete coin flip. In order to define it, we must first define the gradient of leadership and responsiveness threshold.

Let us call the following hash the *round hash*:

$$\mathsf{round\_hash} : H_{X,B} \to [0, 2^{256})$$

$$\mathsf{round\_hash}(e) = \mathsf{hash}(\mathsf{hash}(X), \mathsf{hash}(B), \mathsf{hash}(\mathsf{round}(e)))$$

The *leadership index* of node Y at gossip event e will be its index on the list of all nodes in the network, sorted by XOR-distance from round_hash(e) (the XOR-distance being simply $Y \oplus \mathsf{round\_hash}(e)$).

In order to get the result of the coin flip, we will look for the first gossip event by the node with the leader index 0 that has an aux value in the current step, and take the least significant bit of its hash. There is one caveat, though - the first leader might be dead and never gossip such an event. We can never know for sure because of the asynchronicity assumption.

To get past that hurdle, we use the responsiveness_threshold, which is just an integer function of N. This threshold will denote the number of gossip events caused by responses we will allow ourselves to create before we use another leader - one whose event we see, and who has the lowest leadership index.

Formally, it would look like this:

$$\mathsf{ev\_waited} : H_{X,B} \to \mathbb{N}$$

$$\mathsf{ev\_waited}(e) = \begin{cases} 0 & \text{if } \mathsf{supermajority\_valid\_aux}(e) \\ & \text{and not} \\ & \mathsf{supermajority\_valid\_aux}(p) \\ 1 + \mathsf{ev\_waited}(p) & \text{if } \mathsf{cause}(e) = \mathit{response} \\ \mathsf{ev\_waited}(p) & \text{otherwise} \end{cases}$$

$$\text{where } p = \mathsf{self\_par}(e)$$

The function ev_waited counts how many events passed since the first one where we could theoretically flip the coin. We can use this to check whether the first leader "timed out".

But first, we will define a function that gives us the first event by a node which has an aux value:

$$\mathsf{first\_aux} : \mathcal{N} \times H_{X,B} \to H_{X,B} \cup \{\bot\}$$

$$\begin{aligned} \mathsf{first\_aux}(Y, e) = \ & \text{the event } x \text{ created by Y such that } e \text{ sees } x \\ & \text{and } \mathsf{aux}(x) \neq \bot \text{ and } \mathsf{aux}(\mathsf{self\_par}(x)) = \bot \\ & \text{and } \mathsf{stage}(e) = \mathsf{stage}(x) \text{ if it exists; } \bot \text{ otherwise} \end{aligned}$$

Let us denote the first leader by L. Then the function that tells us if the "timeout" happened will be:

$$\mathsf{leader\_timed\_out} : H_{X,B} \to \{0, 1\}$$

$$\begin{aligned} \mathsf{leader\_timed\_out}(e) = \ & \mathsf{first\_aux}(L, e) = \bot \text{ and} \\ & \mathsf{ev\_waited}(e) > \mathsf{responsiveness\_threshold}(N) \end{aligned}$$

The event that generates the coin will be calculated the following way:

$$\mathsf{coin\_event} : H_{X,B} \to H_{X,B} \cup \{\bot\}$$

$$\mathsf{coin\_event}(e) = \begin{cases} \mathsf{first\_aux}(L, e) & \text{if not } \mathsf{leader\_timed\_out}(e) \\ \mathsf{first\_aux}(Y, e) & \text{if } \mathsf{leader\_timed\_out}(e), \\ & \text{where } Y \text{ is the node with} \\ & \text{the lowest leadership index} \\ & \text{for which } \mathsf{first\_aux}(Y, e) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

Then, the coin flip will be just:

$$\mathsf{coin\_flip} : H_{X,B} \to \{0, 1, \bot\}$$

$$\mathsf{coin\_flip}(e) = \begin{cases} \text{lowest order bit} & \\ \text{of } \mathsf{hash}(\mathsf{coin\_event}(e)) & \text{if } \mathsf{coin\_event}(e) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

We call the result of the coin_flip function a *genuinely flipped concrete coin.* This is all we need to reach consensus on the meta-votes.

2.4.2 Agreement about the next block

Using the above, every node can calculate the results of all meta-elections. Once the results are known, they can be used to determine the next block in the ordered set.

Let us remember that the meta-elections started with a set of observers - a set of events that all strongly see a supermajority of block-votes. The results of the meta-elections tell us which block-votes are to be taken into account.

The properties of meta-elections ensure that all nodes will agree on the considered set of nodes. What we need to do is change that into an agreement on what the next block should be. This is pretty trivial, although we must consider two issues: every node could create multiple block-votes, and every block-vote could see multiple valid blocks.

To counter the first issue, we can just take the earliest block-vote created by a given node. The events created by a single node form a linear sequence, so the earliest one is well-defined. This narrows the considered set down to a single block-vote per node.

The next step is to choose a valid block among potentially multiple ones seen by the chosen block-vote. To do that, we can take the lexicographically first one, or use really any method that will always choose the same element of a set.

Once we have one vote on a block per node, we just count them and the next agreed block will be the one with the most votes. Any ties can be broken again by lexicographic ordering, or some other method.

This completes the description of the algorithm. The next section will prove that it is correct, that is, that it provides robust consensus in an asynchronous setting, and in the presence of Byzantine faults.

3 Proof of correctness

Let us begin by stating two important properties of the gossip graph.

Definition 3.1. We call two gossip graphs *consistent* iff for every gossip event x that is present in both graphs, both contain the same set of ancestors of x with the same sets of edges between them.

Lemma 3.1. *All nodes in the network have consistent gossip graphs.*

Lemma 3.2. *If a pair of gossip events* (x, y) *is a fork, and another gossip event* z *strongly sees* x*, then no other gossip event in a consistent graph can strongly see* y*.*

We won't prove the above lemmas - they have been proved in [1] (as Lemma 5.11 and 5.12, respectively).

Let us now prove some properties of our approach stemming from it being an adaptation of [2].

Lemma 3.3 (Progress). *If a correct node created a gossip event in stage s, every other correct node will eventually create an event in stage s as well.*

Proof. Step numbers are based on seeing a supermajority of some events - either events seeing a valid block (for observers, stage 0), or events in the previous stage having valid `aux` values (stage greater than 0). Assume there is event e in stage s created by a correct node - it means that it sees a set of events that allowed it to proceed to stage s. A correct node will continue gossiping, so every other correct node will eventually learn of e and create a descendant of e.

A descendant of e sees everything that e sees (provided that e is not a part of a fork - but its creator is correct, so it is not). If e was in stage 0, any descendant will thus automatically be in stage at least 0.

Assume that the lemma is true for stage $s-1$. e has a self-ancestor in stage $s-1$, which means that every correct node will eventually create an event in stage $s-1$. Any later event which has e as an ancestor will thus have an event in stage $s-1$ as a self-ancestor, and will see events allowing it to progress to stage s. Thus, the lemma is also true for s. By induction, the proof is complete. □

Lemma 3.4. *The probability that the genuinely flipped concrete coin is common and pseudorandom is $> \frac{2}{3} - \varepsilon$, with an arbitrarily small ε when the responsiveness threshold is sufficiently high, and the required threshold value is logarithmic in N.*

Proof. First, note that the node with leadership index 0 is common and pseudorandom. It is common, as the nodes share the list of the nodes in the network, the last decided block in the ordered set and the round number. It is pseudorandom, as it is based on a cryptographic hash of values outside the nodes' control.

With probability $> \frac{2}{3}$, the node with leadership index 0 will be honest. In such a case, it gossips its events honestly to random nodes, so that correct nodes will receive the coin event before "timeout" with probability p, which is close to 1.

The probability of the coin being common and random is strictly larger than the probability of the event described above. Such an event's probability, on the other hand, is $> \frac{2}{3}p$. As long as $p > 1 - \frac{3}{2}\varepsilon$, this is $> \frac{2}{3} - \varepsilon$. $p > 1 - \frac{3}{2}\varepsilon$ can be ensured by choosing the right responsiveness threshold. In fact, p can be arbitrarily close to 1 with a large enough responsiveness threshold.

Since gossip from correct nodes is expected to reach everyone in $O(\log N)$ exchanges, the minimal responsiveness threshold will also be logarithmic in N. □

Lemma 3.5. *If all correct nodes created events in stage $s \equiv 2 \pmod 3$ (step 2), and no such event decided a value, then the estimates of their events in the next stage will be in agreement with probability $> \frac{1}{3} - \varepsilon'$ for arbitrarily small ε'.*

Proof. The estimate in a stage after the coin stage is based on the auxiliary values seen by events in the coin stage. There are five possible situations here:

- All correct nodes' events in stage $s+1$ see a supermajority of $true$ values from stage s - estimates agree with probability 1.
- All correct nodes' events in stage $s+1$ see a supermajority of $false$ values from stage s - estimates agree with probability 1.

- Some correct nodes' events in stage $s+1$ see a supermajority of $true$ values from stage s; the others throw a coin, which is common and random with probability $> \frac{2}{3} - \varepsilon$, and the result is $true$ with probability $\frac{1}{2}$ - estimates agree with probability $> \frac{1}{3} - \varepsilon'$ (where $\varepsilon' = \frac{1}{2}\varepsilon$).

- Some correct nodes' events in stage $s + 1$ see a supermajority of $false$ values from stage s; the others throw a coin, which is common and random with probability $> \frac{2}{3} - \varepsilon$, and the result is $false$ with probability $\frac{1}{2}$ - estimates agree with probability $> \frac{1}{3} - \varepsilon'$.

- No correct nodes' events in stage $s + 1$ see a supermajority of any value in stage s; all of them throw a coin, which is common and random with probability $> \frac{2}{3} - \varepsilon$ - estimates agree with probability $> \frac{2}{3} - \varepsilon$.

The total probability of agreement is a weighted average of five values, all greater than $\frac{1}{3} - \varepsilon'$, which means the total probability is also greater than this value. □

Lemma 3.6. *If all correct nodes' first events in stage s had* estimates $= \{v\}$, *their first events in stage $s + 1$ will also have* estimates $= \{v\}$.

Proof. If all correct nodes only estimate v, there is no way for any event to see even $\frac{N}{3}$ of estimates for $\neg v$ - so no event by a correct node will have it in its estimates in stage s.

For a value to be an element of bin_values, there must be a supermajority of events estimating that value. Because of the above, the only value that can have a supermajority is v. Thus, every event with nonempty bin_values will have it equal to $\{v\}$. Hence, every event with an aux value will have it equal to v.

In order to proceed to the next stage, an event has to see a supermajority of valid aux values. No event can have a value other than v as aux in stage s, so there will always be a supermajority for v. Depending on the stage number, this can either lead to deciding v, or estimating v in stage $s + 1$. Either way, the agreement will still hold. □

Lemma 3.7. *If all correct nodes' first events in round r had* estimates $= \{v\}$, *they will all decide v within round r.*

Proof. No matter what malicious nodes do, there is less than a third of them, so no event by a correct node will have $\neg v$ in estimates (by definition of the est function). This means that for bin_values of an event to be non-empty, it must see a supermajority of estimates for v, as there will never be a supermajority for $\neg v$.

The above means that no correct node will add $\neg v$ to bin_values, so all of them will eventually create an event with aux $= v$. This means there will be a supermajority of events by different creators with aux $= v$, which will make the correct nodes either decide at step 0 (if $v = 1$), or estimate 0 for the next step and decide then. □

Theorem 3.8 (Binary Byzantine Consensus)**.** *The algorithm for calculating meta-election results presented in this paper satisfies the general properties of a Byzantine fault tolerant consensus algorithm:*

- Validity - *if a correct node decides on a value, it has been proposed by a correct node.*

- Agreement - *if a correct node decides on a value, all correct nodes decide on that value.*

- Integrity - *once a correct node decides on a value, it never decides on another value.*

- Termination - *all correct nodes eventually decide with probability 1.*

Validity. We will prove an equivalent statement: that if initially all correct nodes propose v, then all correct nodes will decide v. Since v is a binary value, a node can only decide a value not proposed by a correct node if all correct nodes propose v, and the node decides $\neg v$. Thus, deciding v when all correct nodes propose v is equivalent to always deciding on a value proposed by a correct node.

If all correct nodes propose v, they will all put v in their estimates. By Lemma 3.7, they will all decide v within the first round. □

Agreement. Assume there is an event e created by a correct node which was able to decide a value v. It means that $\mathsf{step}(e) = 0$ or $\mathsf{step}(e) = 1$ (there would be no decision otherwise). This event must have seen a supermajority of events with $\mathsf{aux} = v$, which means there was no supermajority for $\neg v$. Thus, if a correct node has seen a supermajority in this stage, it must have been for v, so it would decide v. If it wasn't, it would estimate v for the next stage, which means there will be agreement at the start of the next stage. Following Lemma 3.6, this agreement will propagate to the end of the stage and the next stage, until everyone decides v. □

Integrity. Once an event e created by a correct node decides on a value v, all later events created by that node will have event e as an ancestor. Following the definition of meta_election, all later events will also decide v. □

Termination. By Lemma 3.3, if a correct node creates an event in stage s, then every correct node eventually creates an event in stage s. This means there will be events by $> \frac{2}{3}N$ correct nodes, which will eventually be seen by every correct node. Every such event will have non-empty estimates. It is not possible for both 0 and 1 to be estimated by $< \frac{N}{3}$ events by different correct nodes, so at least one of those values will eventually be an element of estimates of every correct node's event.

Eventually, the events with agreeing estimates will all be seen by an event created by every correct node. Hence, every correct node will eventually create an event with non-empty bin_values, and so an aux value.

The events with aux values will eventually be seen by every correct node's event, which means every correct node will eventually either decide or progress to the next stage.

By Lemma 3.5, if no node decided in step 0 or 1, they will all agree after step 2 with probability $> \frac{1}{3} - \varepsilon'$. By Lemma 3.7, this means they will decide in the next round with probability $> \frac{1}{3} - \varepsilon'$. Thus, the probability of no agreement in round r is $< \left(1 - \frac{1}{3} + \varepsilon'\right)^4 < \left(\frac{2}{3} + \varepsilon'\right)^r$. This tends to 0 as r increases, so the nodes will eventually decide with probability 1. □

The above theorem proves that our algorithm will reach agreement about every single meta-election in a Byzantine fault tolerant way. This is not the end, though - we also need to prove that meta-elections lead to agreement about the next block in the ordered set. The proof of that is presented below.

Lemma 3.9. *If the result of a meta-election is v, there have been at least $\frac{N}{3}$ meta-votes for v.*

Proof. Assume there have been less than $\frac{N}{3}$ meta-votes for v and v has been decided. When nodes that initially meta-voted v create an event that sees a supermajority of meta-votes, this supermajority must contain at least $\frac{N}{3}$ votes for $\neg v$ - so their estimates will contain $\neg v$. On the other hand, no node that meta-voted $\neg v$ can ever create an event that will see at least $\frac{N}{3}$ estimates for v, so they won't add v to estimates.

Due to the above, any supermajority among the estimates must be for $\neg v$. Any event with non-empty bin_values can thus only have $\neg v$ in this set, which means that all valid aux values will also be $\neg v$, which will lead to a decision on $\neg v$ within two stages.

This is a contradiction. Such a situation is impossible, which proves the lemma. □

Lemma 3.10. *The set of nodes for which the result of meta-election is true is always non-empty.*

Proof. Assume all meta-elections resulted in $false$. By Lemma 3.9, at least $\frac{N}{3}$ nodes meta-voted $false$ for every node, so there have been at least $\frac{N^2}{3}$ meta-votes for $false$.

On the other hand, by definition of an observer, every node voted $true$ for more than $\frac{2}{3}N$ nodes - so there have been more than $\frac{2}{3}N^2$ meta-votes for $true$, which leaves less than $\frac{N^2}{3}$ meta-votes for $false$ (there are N^2 meta-votes in total: N nodes meta-vote in N meta-elections).

This is a contradiction, which proves the lemma. □

Theorem 3.11 (Byzantine Consensus)**.** *The algorithm for calculating the next block presented in this paper satisfies the general properties of a Byzantine fault tolerant consensus algorithm:*

- Validity - *if a correct node decides on a value, it has been proposed by a correct node.*
- Agreement - *if a correct node decides on a value, all correct nodes decide on that value.*
- Integrity - *once a correct node decides on a value, it never decides on another value.*
- Termination - *all correct nodes eventually decide with probability 1.*

Validity. Assume a correct node decided that the next block should be a block B containing a network event E. For such a situation to happen, there must have been a block-vote for B, and such a block-vote must have seen a supermajority of votes for E. A supermajority will always contain a correct node, so a correct node must have voted for E. □

Agreement. Assume that a correct node decided the next block B. Since a decision has been reached, this means that there is consensus about the meta-votes, so every correct node will have chosen the same nodes' block-votes.

For any observer to meta-vote *true* on a node, it must have strongly seen a block-vote by that node. By Lemma 3.2, even if that node created a fork, if any other observers also voted *true* on that node, they must have strongly seen block-votes on the same fork. Thus, we can consider block-votes by all elected nodes to form linear histories - which will be seen the same way by all nodes by Lemma 3.1.

In a linear history, the earliest block-vote is well-defined. Also, because all correct nodes see the same history, they will all choose the same block-vote as the earliest. If the block-vote votes for multiple blocks, all correct nodes will use the same tie-breaker algorithm and choose the same single one. Thus, all correct nodes will gather the same set of votes, and because they use the same voting rules, decide the same block as the next one. □

Integrity. By construction of the algorithm, once the next block has been decided, it is appended to the ordered set and no other block can be decided in its place. □

Termination. For every network event, we expect all correct nodes to eventually vote for it. This means there will be a supermajority of votes for every event, which will eventually be seen by some other events - block-votes - created by each correct node. The block-votes will eventually be strongly seen by other events created by each correct node - the observers. Once there is a supermajority of observers, we start the binary agreement algorithm, which will terminate by Theorem 3.8. After binary agreement terminates, because the set of voters for the next block is non-empty (by Lemma 3.10), the next block is already determined - so the agreement about the next block also terminates. □

4 Conclusions

A new consensus algorithm has been presented, building upon some previous achievements in this field ([1], [2], [3]), but combining their features in a novel way. It is asynchronous (as in [1] and [2]), uses a gossip graph (as in [1] and [5]), and generates randomness, which is necessary for asynchronous consensus [6], using a concrete coin (as in [3]). We believe this approach will be useful in numerous applications, one of which is the SAFE Network.

References

[1] Baird L. *The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance*, Swirlds Tech Report SWIRLDS-TR-2016-01 (2016)

[2] Mostefaoui A., Hamouna M., Raynal M. *Signature-Free Asynchronous Byzantine Consensus with* $t < n/3$ *and* $O(n^2)$ *Messages*, ACM PODC (2014)

[3] Micali S. *Byzantine Agreement, Made Trivial*, (2018)

[4] Miller A., Xia Y., Croman K., Shi E., Song D. *The Honey Badger of BFT Protocols*, CCS (2016)

[5] Team Rocket *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*, (2018)

[6] Fischer M. J., Lynch N. A., Paterson M. S. *Impossibility of Distributed Consensus with One Faulty Process*, JACM Vol. 32 Issue 2 (April 1985)